Année universitaire	**2017-2018**		
Département	**Informatique**	**Année**	**M2**
Matière	**Machine Learning**		
Intitulé TD/TP :	**TP1 Machine Learning sous Python**		
Durée	**12h**		

Ce TP est à réaliser seul ou en binôme (trinômes interdits). Il est à rendre pour le 22/12/2017, 18h00. Il faut rendre votre notebook Python avec toutes les parties bien séparées et commentées. Ne pas oublier de mentionner les deux étudiants du binôme dans le nom du fichier qui sera sous la forme : TP_ML_Nom1_Prenom1_Nom2_Prenom2.ipynb. Nom1_Prenom1 est le nom et le prénom du premier membre du binôme, Nom2_Prenom2 est le nom et le prénom du second membre du binôme.

Le non-respect de ces consignes pourra être sanctionné dans la note de ce TP.

Dans ce TP, vous allez expérimenter des algorithmes de traitement de données dont certaines ont été abordées en Cours pour répondre à différents problèmes d'apprentissage automatique avec le langage **Python**.

Python est un langage de programmation très polyvalent et modulaire, qui est utilisé aussi bien pour écrire des applications comme YouTube, que pour traiter des données scientifiques. Par conséquent, il existe de multiples installations possibles de Python. L'utilisateur débutant peut donc se sentir dérouté par l'absence d'une référence unique pour Python scientifique. Le plus simple pour ce TP est d'installer, la suite scientifique Anaconda développée par l'entreprise Continuum (http://continuum.io/downloads.html). Anaconda rassemble tout le nécessaire pour l'enseignement de Python scientifique: le langage Python et ses modules scientifiques. Sur le plan des packages Python, vous allez utiliser **Scikit-learn**. Cette libraire montre dans cette situation tout son intérêt. La plupart des techniques récentes d'apprentissage sont en effet expérimentées avec Scikit-learn et le plus souvent mises à disposition de la communauté scientifique.

Pour plus de détails concernant :

- le langage Python vous pouvez aller sur le site suivant : http://www.python-course.eu/index.php
- la librairie Scikit-learn vous pouvez aller sur le site suivant : http://scikit-learn.org

Pour lancer le notebook Python, il faut taper la commande **jupyter notebook** dans le dossier TP1. Une fenêtre va se lancer dans votre navigateur pour ouvrir l'application Jupyter. Créer un nouveau notebook Python et taper le code suivant dans une nouvelle cellule :

```
import numpy as np
np.set_printoptions(threshold=np.nan)
import pandas as pd
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

I. Apprentissage non supervisé : Segmentation et Visualisation

Le fichier **"villes.csv"** comporte 32 villes françaises décrites par les températures moyennes dans les 12 mois de l'année.

L'objectif dans cette partie est de représenter graphiquement le plus d'informations possibles contenues dans ce fichier de données et de déceler une éventuelle segmentation topologique des villes.

1. Importer ce jeu de données avec la librairie **pandas** (*c.f.* ***read_csv***)

```
data = pd.read_csv('./villes.csv', sep=';')
X = data.ix[:, 1:13].values
labels = data.ix[:, 0].values
```

2. Réaliser une Analyse en Composantes Principales (module **PCA** de Scikit-learn) sur ce jeu de données centrées réduites (**StandardScaler**)
   - Quel est le nombre d'axes à retenir pour conserver un minimum de 70% de l'information représentée dans le nuage initial.
   - Donner une interprétation des deux premiers axes principaux.
   - En suivant le code suivant, donner une visualisation graphique des villes projetées dans le plan principal.

   X_pca étant la matrice des données transformées par l'ACP, **labels** étant le vecteur contenant le nom des instances (ici les villes).

```
import matplotlib
plt.scatter(X_pca[:, 0], X_pca[:, 1])
for label, x, y in zip(labels, X_pca[:, 0], X_pca[:, 1]):
  plt.annotate(label, xy=(x, y), xytext=(-0.2, 0.2), textcoords='offset points')
plt.show()
```

3. Faire de même pour le fichier **"crime.csv".** Il s'agit des statistiques de criminalité dans 50 états américains. Dans chaque état, sept types de crimes ou délits sont repérés par leurs nombres annuels de faits constatés rapportés sur 100 000 habitants : meurtres(Meutre), enlèvements(Rapt), vols avec violence(Vol), agressions(Attaque), viol (Viol), vols peu importants (Larcin), vols de voitures (Auto_Theft).

La classification (les Anglos-saxons parlent de clustering) est l'opération statistique qui consiste à regrouper des objets (individus ou variables) en un nombre limité de groupes, les classes (ou segments, ou clusters) , qui ont deux propriétés. D'une part, ils ne sont pas prédéfinis par l'analyste mais découverte au cours de l'opération, contrairement aux classes du classement. D'autre part, les classes de la classification regroupent les objets ayant des caractéristiques similaires et séparent les objets ayant des caractéristiques différentes (homogénéité interne et hétérogénéité externe), ce qui peut être mesuré par des critères telle l'inertie interclasse et l'inertie intraclasse.

Nous allons étudier dans la suite deux approches de clustering (*k-moyennes* **"KMeans"** et la *classification Ascendante Hiérarchique* **"AgglomerativeClustering"** du package sklearn.cluster) que nous allons appliquer sur le jeu de données des villes.

4. Appliquez la procédure **KMeans** sur ce jeu de données pour obtenir **3** clusters
   - Donner une visualisation graphique des villes projetées dans le plan principal. Les villes de chaque cluster devraient avoir une couleur différente des villes des autres clusters (voir code ci-dessous).

   X_pca étant la matrice des données transformées par l'ACP, **labels** étant le vecteur contenant le nom des instances (ici les villes), **clustering** étant le clustering obtenu.

```
colors = ['red','yellow','blue','pink']
plt.scatter(X_pca[:, 0], X_pca[:, 1], c= clustering, cmap=matplotlib.colors.ListedColormap(colors))
for label, x, y in zip(labels, X_pca[:, 0], X_pca[:, 1]):
  plt.annotate(label, xy=(x, y), xytext=(-0.2, 0.2), textcoords='offset points')
plt.show()
```

5. Appliquez la procédure **AgglomerativeClustering** sur ce jeu de données pour obtenir trois clusters avec différentes méthodes d'agrégation (il faut essayer **ward** et **average**).
   - Donner à chaque fois une visualisation graphique des villes projetées dans le plan principal. Les villes de chaque cluster devraient avoir une couleur différente des vilels des autres clusters.

Nous allons maintenant déterminer la meilleure partition (nombre de clusters) pour la méthode **KMeans**. Pour cela, nous allons utiliser deux critères **"Silhouette index"** (**metrics.silhouette_score** de scikit-learn) et de **"Dunn index"** (fonction définie dans le fichier **"dunn.txt"**).

6. Utiliser ces deux indices dans une boucle de 4 itérations au maximum (voir code ci-dessous). Les 4 itérations correspondent aux 4 partitions possibles *i.e.* en 2, 3, 4 et 5 classes issues de **KMeans**. Déduire la meilleure partition qui correspond à un indice maximal pour chacun des deux indices **Silhouette** et **Dunn**.

```
from sklearn import metrics
for i in np.arange(2, 6):
    clustering = KMeans(n_clusters=i).fit_predict(X)
    print(dunn(clustering, euclidean_distances(X, X)))
    print(metrics.silhouette_score(X, clustering,metric='euclidean'))
    print()
```

II. Apprentissage supervisé : Feature engineering et Classification

L'objectif dans cette partie est de faire une étude comparative entre plusieurs algorithmes d'apprentissage supervisé sur un jeu de données de *crédit scoring*. Pour plus d'informations sur ce jeu de données, vous pouvez visiter le lien suivant (https://archive.ics.uci.edu/ml/datasets/Credit+Approval).

Le fichier **"credit.data"** comporte 688 instances décrites par 15 variables caractéristiques (6 numériques, 9 catégorielles) et la variable à prédire "classe" (la dernière colonne du fichier) de nature nominale possédant un nombre fini de valeurs (ici deux valeurs "+" et "–"). Il ne s'agit pas d'une tâche de régression, mais de classification. Les exemples de ce jeu de données représentent des personnes (positifs et négatifs) pour lesquels un crédit a été accordé ou non.

1. Dans un premier temps nous allons considérer que les caractéristiques continues (numériques) de ce jeu de données sans les données manquantes.

   - ***Chargement des données et préparation*** **:**
     - Importer ce jeu de données avec la librairie **pandas** (*c.f.* ***read_csv***)
     - Transformer votre jeu de données issue de **pandas** qui sera de type **Data Frame** en **numpy Array** (*c.f.* ***values***) et séparer ensuite les variables caractéristiques de la variable à prédire (target) en deux tableaux différents.
     - Créer un sous ensembles de vos données en gardant que les variables numériques et en remplaçant les valeurs manquantes (?) par des **nan**. N'oublier de typer votre tableau en type **float** (*c.f.* ***astype***).
     - Supprimer les individus dans vos données contenant des **nan** sur au moins une variable.
     - Analyser les propriétés de vos données : taille de l'échantillon (*c.f.* ***shape***), nombre d'exemples positifs et négatifs (*c.f.* ***hist***).
     - Binariser votre target (+ en 1 et – en 0).

   - ***Apprentissage et évaluation de modèles*** **:** Utiliser ensuite sur votre jeu de données les algorithmes d'apprentissage supervisé suivants :
     - NaiveBayesSimple
     - Un arbre CART
     - Un arbre ID3
     - Decision Stump
     - MultilayerPerceptron à deux couches de tailles respectives 20 et 10 par exemple
     - k-plus-proches-voisins avec k=5 par exemple
     - Bagging avec 50 classifieurs par exemple
     - AdaBoost avec 50 classifieurs par exemple
     - Random Forest avec 50 classifieurs par exemple
     - Etc.

     Si vous ne connaissez pas le fonctionnement précis de l'un de ces algorithmes, n'hésitez pas à consulter la documentation de Scikit-learn. Attention, la plupart de ces algorithmes ont des paramètres qu'il vous faudra prendre en compte lors de vos expérimentations. Vous en choisirez quelques-uns que vous ferez varier afin d'observer l'effet pratique de ces paramètres sur les résultats obtenues (par exemple ***k*** pour les ***k-plus-proches-voisins*** ou le nombre d'arbres dans un Random Forest).
     L'objectif est à présent de comparer les résultats obtenus à l'aide des différents algorithmes donnés ci-dessus sur votre jeu de données. Ces comparaisons s'appuieront sur :
     - l'estimation de l'***accuracy*** et de l'***AUC*** (Aire sous la courbe ROC) par 10 fold cross-

validation (*c.f.* ***cross_val_score***). Il faut afficher la moyenne et l'écart type.

- l'estimation aussi par 5 fold cross-validation du critère qu'il vous semble le plus pertinent entre le ***rappel*** et la ***précision*** pour cette tâche du crédit scoring. Il faut afficher aussi la moyenne et l'écart type.
- le temps d'exécution de l'algorithme d'apprentissage (*c.f.* ***time.time()***)

Note :

Afin de mieux comparer plusieurs algorithmes sur une même validation croisée, il est préférable de passer par un dictionnaire dans lequel vous mettez la liste des algorithmes à comparer et d'utiliser le code suivant :

```
clfs = {
  'RF': RandomForestClassifier(n_estimators=50),
  'KNN': KNeighborsClassifier(n_neighbors=10),
 liste à compléter
}
kf = KFold(n_splits=10, shuffle=True, random_state=0)
for i in clfs:
   clf = clfs[i]              //clf correspond au ième algorithme dans votre dictionnaire clfs.
   cv_acc = cross_val_score(clf, X, Y, cv=kf) //pour le calcul de l'accuracy
   print("Accuracy for {0} is: {1:.3f} +/- {2:.3f}".format(i, np.mean(cv_acc), np.std(cv_acc)))
```

Créer une fonction Python ***run_classifiers*** qui permettra de lancer la comparaison des algorithmes de classification supervisée et qui prendra en paramètre une dictionnaire **clfs**, le tableau de données caractéristiques **X** et la target **Y**. Elle sera utilisée dans la suite de ce TP. (c.f. ***def***). Exécuter cette fonction et interpréter les résultats obtenus.

- ***Normalisation des variables continues*** **:** Certains algorithmes d'apprentissage supervisé fonctionneront mieux si les données sont normalisées (centrées autour de 0) pour que toutes les variables caractéristiques auront le même poids dans la phase d'apprentissage. Utiliser le module ***StandardScaler*** de Scikit-learn pour normaliser vos données. Vous pouvez également tester le module ***MinMaxScaler***. Exécuter votre fonction ***run_classifiers*** sur vos données une fois normalisées. Interpréter les résultats obtenus en les comparant avec ceux de la question précédente.

- ***Création de nouvelles variables caractéristiques par combinaisons linéaires des variables initiales*** **:** Il est parfois utile pour certains classifieurs de faire une réduction de dimensions sur les données afin de déceler et créer certaines combinaisons linéaires dans les variables descriptives et augmenter ainsi le pouvoir discriminant du classifieur. Appliquer une **ACP** (module ***PCA*** de Scikit-learn) sur vos données et garder les *k* premières nouvelles dimensions en les concaténant ou pas à vos données normalisées de l'étape précédente (la valeur de *k* est à choisir pour garder au moins 70% de l'information). Exécuter votre fonction ***run_classifiers*** sur vos nouvelles données. Que se passe t-il ?

- ***Sélection de variables*** **:** Même si vous utilisez le meilleur algorithme d'apprentissage, la présence de variables bruités pourra avoir un impact négatif sur les résultats d'apprentissage. La sélection de variables est un processus très important en apprentissage supervisé. Il consiste à sélectionner le sous-ensemble de variables les plus pertinentes à partir de la série de variables candidates permettant de mieux expliquer et prédire votre target. Dans la suite, vous allez utiliser la méthode **Random Forest** de Scikit-learn pour déterminer quelles sont les meilleures variables pour prédire si on doit attribuer ou non le crédit à une personne.

  Ainsi, il faut afficher un histogramme des importances des variables. Déterminer ensuite le nombre de variables à garder et exécuter ensuite votre fonction ***run_classifiers*** sur vos données. Comparer les résultats avant et après sélection de variables.

```
clf = RandomForestClassifier()
```

```
clf.fit(X2, Y2)
importances=clf.feature_importances_
sorted_idx = np.argsort(importances)[::-1]

features =np.arange(1, X2.shape[1])

padding = np.arange(X2.size/len(X2)) + 0.5
plt.barh(padding, importances[sorted_idx], align='center')
plt.yticks(padding, features[sorted_idx])
plt.xlabel("Relative Importance")
plt.title("Variable Importance")
plt.show()
```

1. Nous allons maintenant considérer la totalité de la base originale comportant les 15 variables continues et catégorielles mais aussi les données manquantes.

   - ***Traitement de données manquantes*:** La non utilisation des données manquantes peut impacter la phase d'apprentissage suite à la perte d'information susceptible d'être pertinente et/ou informative, surtout quant la proportion des données manquantes dans l'échantillon est forte. Pour traiter les données manquantes deux méthodologies sont possibles : Soit (1) d'utiliser une technique d'imputation de valeurs manquantes, ou (2) non en intégrant des indicateurs de données manquantes dans l'échantillon par l'ajout par exemple d'une modalité à votre variable catégorielle incomplètement remplie (remplacer par exemple le '?' dans une variable sexe avec deux modalités 'Femme' et 'Homme' par 'Inconnu'). Dans la suite, vous allez utiliser la première méthodologie. Pour imputer des données manquantes, différentes stratégies sont possibles, certaines sont supervisées (utilisant la variable target pour remplir les valeurs manquantes dans les autres variables) et d'autres non supervisées. Pour plus de détails considérant ces approches, les liens suivants pourront vous intéresser http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-idm.pdf et http://cybertim.timone.univ-mrs.fr/Members/rgiorgi/DossierPublic/Enseignement/TraitementNA-PDF-RG/docpeda_fichier.
     Imputer les valeurs manquantes dans votre jeu de données en utilisant les stratégies non supervisées suivantes du module ***Imputer*** de Scikit-learn (voir code ci-dessous) :
     - ***mean*** pour les variables continues
     - ***most_frequent*** pour les variables catégorielles

     Attention : Imputer ne considère que des données sous format de **float**. Il faut d'abord transformer les colonnes catégorielles en valeurs numériques (par exemple ['a', 'b', 'a'] en [1,2,1]. Il faut utiliser pour cela le code suivant sur votre tableau de données avec les données catégorielles (ici c'est X_cat) :

Pour les variables catégorielles

```
X_cat = np.copy(X[:, col_cat])
for col_id in range(len(col_cat)):
   unique_val, val_idx = np.unique(X_cat[:, col_id], return_inverse=True)
   X_cat[:, col_id] = val_idx

imp_cat = Imputer(missing_values=0, strategy='most_frequent')
X_cat[:, range(5)] = imp_cat.fit_transform(X_cat[:, range(5)])
```

Pour les variables numériques

```
X_num = np.copy(X[:, col_num])
X_num[X_num == '?'] = np.nan
X_num = X_num.astype(float)

imp_num = Imputer(missing_values=np.nan, strategy='mean')
X_num = imp_num.fit_transform(X_num)
```

- ***Traitement de variables catégorielles*** **:** Pour pouvoir utiliser les variables catégorielles dans les algorithmes d'apprentissage supervisé de votre fonction ***run_classifiers***, une solution consiste à transformer chaque variable catégorielle avec ***m*** modalités en ***m*** variables binaires dont une seule sera active. Pour cela utiliser le module ***OneHotEncoder*** de Scikit-learn pour encoder les 9 variables catégorielles de votre jeu de données.

  X_cat_bin = OneHotEncoder().fit_transform(X_cat).toarray()

- ***Construction de votre jeu de données*** **:** Construire votre jeu de données en concaténant à la fois les données catégorielles transformées et les données continues (Ne pas oublier de normaliser vos données numériques et de rajouter les variables issues de l'ACP. L'objectif étant de bien préparer les données originales afin d'obtenir le meilleur jeu de données sur lequel vos classifieurs seront appris.

- ***Sélection de variables*** **:** Afficher un histogramme des importances des variables.

III. *Apprentissage supervisé sur des données textuelles* : Feature engineering et Classification

L'objectif dans cette partie est de faire une étude comparative entre plusieurs algorithmes d'apprentissage supervisé sur un jeu de données textuelles de SMS ("SMSSpamCollection.data") pour prédire si un message est un spam ou pas. Pour plus d'informations sur ce jeu de données, vous pouvez visiter le lien suivant (https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection).
Chaque ligne dans ce fichier correspond à la classe d'un SMS ('***spam'*** ou '***ham***') et le texte du SMS. Avant de pouvoir appliquer votre fonction ***run_classifiers*** sur ce jeu de données, il faut faire un pré-traitement de ces données afin de transformer le texte de chaque SMS en un vecteur de données (on parle ici du ***Text Mining***). Vous allez utiliser pour cela la représentation *bag-of-words*. Une fois les données chargées, les étapes suivantes doivent être exécutées dans l'ordre en suivant la documentation dans la page officielle du package ***sklearn.feature_extraction.text*** de Scikit-learn (http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction) :

- ***CountVectorizer*** **:** pour splitter chaque texte en différents mots clés (termes), supprimer les mots clés vides (stopwords) et calculer la matrice de co-occurences. Il est possible de garder que les mots clés les plus fréquents. Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et interpréter les résultats obtenus.

- ***Tf–idf term weighting*** **:** une mesure statistique utilisée pour la normalisation et la pondération de l'importance d'un terme contenu dans un document, relativement à toute la collection des documents (ici les SMS). Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et interpréter les résultats obtenus en les comparant à ceux obtenus à l'étape précédente.

- ***TruncatedSVD :*** une méthode de réduction de dimensions pour les matrices très creuses (sparses en anglais) qui permettra d'améliorer la représentation vectorielle des textes des SMS par une indexation sémantique latente. Cette dernière permet d'établir des relations entre un ensemble de documents et les termes qu'ils contiennent, en construisant des "concepts" liés aux documents et aux termes. Elle permettra entre autre de résoudre les problèmes de synonymie (plusieurs mots avec un seul sens) et de polysémie (un seul mot avec plusieurs sens). La SVD est bien documentée en Scikit-learn. Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et interpréter les résultats obtenus en les comparant à ceux obtenus à l'étape précédente.

IV. Détection d'anomalies

La détection d'anomalies (dite aussi détection d'outliers) est une tâche de l'apprentissage automatique qui consiste à déceler dans les données, les instances (individus) ayant un comportement différent (inhabituel) des autres instances de la base dites normales. Dans le cas de la détection des fraudes par exemple, toute dépense très étrange par carte de crédit est suspecte. Les variations possibles sont si nombreuses et les exemples de

formation si rares, qu'il n'est pas possible de savoir à quoi ressemble une activité frauduleuse ou un incident. L'approche de la détection des anomalies consiste simplement à apprendre à quoi ressemble l'activité normale (à l'aide d'un historique de transactions supposées non-frauduleuses) et d'identifier tout ce qui est très différent. Différentes algorithmes ont été proposées dans la littérature dont certaines sont supervisées et d'autres non supervisées. Pour plus de détails considérant ces approches, vous pouvez vous référer vers les liens suivants : https://perso.telecom-paristech.fr/~goix/nicolas_goix_osi_presentation.pdf et https://en.wikipedia.org/wiki/Anomaly_detection

Sickit-learn propose différentes approches pour la détection d'anomalies (http://scikit-learn.org/stable/modules/outlier_detection.html). Nous nous intéressons ici à une approche non supervisée appelée **Isolation Forest** (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html). Nous allons appliquer cette approche sur trois jeux de données **"mouse.txt"** (https://elki-project.github.io/datasets/), le fichier **"creditcard.csv"** du challenge Kaggle (https://www.kaggle.com/dalpozz/creditcardfraud) et le fichier des SMS de l'exercice précédent.

1. Sur la base de données Mouse

Ce fichier contient 500 instances décrites par deux variables x1 et x2 représentant des points de la tête de Mickey Mouse. Les 10 dernières instances du fichier sont aberrantes (outliers).

- Télécharger ce jeu de données et analyser le.
- Donner une représentation graphique des données (***matplotlib.pyplot***).
- Appliquer la technique Isolation Forest pour détecter les anomalies dans votre jeu de données.
- Modifier votre représentation graphique précédente pour visualiser les données aberrantes.

2. Sur le jeu de données des cartes de crédits

L'objectif dans cette partie est de déceler les fraudes dans les transactions de cartes bancaires. Pour plus d'informations sur ce jeu de données, vous pouvez visiter le lien suivant (https://www.kaggle.com/dalpozz/creditcardfraud).

Ce jeu de données est très déséquilibré avec seulement 0.172% de fraudes (492 fraudes sur 284 807 transactions). A partir de ce jeu de données, garder aléatoirement 5000 transactions de cartes normales (***classe 0***) et toutes les transactions aberrants (***classe 1***). On pourra également travailler sur la totalité des données.

- Préparer ce jeu de données (ne pas utiliser la variable Time).
- Appliquer la technique Isolation Forest pour détecter les anomalies dans votre jeu de données.
- Retourner la matrice de confusion et analyser les instances aberrantes détectées.

3. Sur le jeu de données des SMS

A partir du jeu de données des SMS transformées (représentation SVD) de l'exercice précédent, garder 50% des SMS normaux (***ham***) et 20 SMS aberrants (***spam***). L'objectif est d'avoir un data set où la majorité des données sont normales (***ham***) afin d'avoir une meilleure approche de détection d'anomalies.

- Préparer ce jeu de données.
- Appliquer la technique Isolation Forest pour détecter les anomalies dans votre jeu de données.
- Retourner la matrice de confusion et analyser les instances aberrantes détectées.